STRUTTURE DATI
LISTE CONCATENATE DOPPIE
LISTE CONCATENATE CIRCOLARI

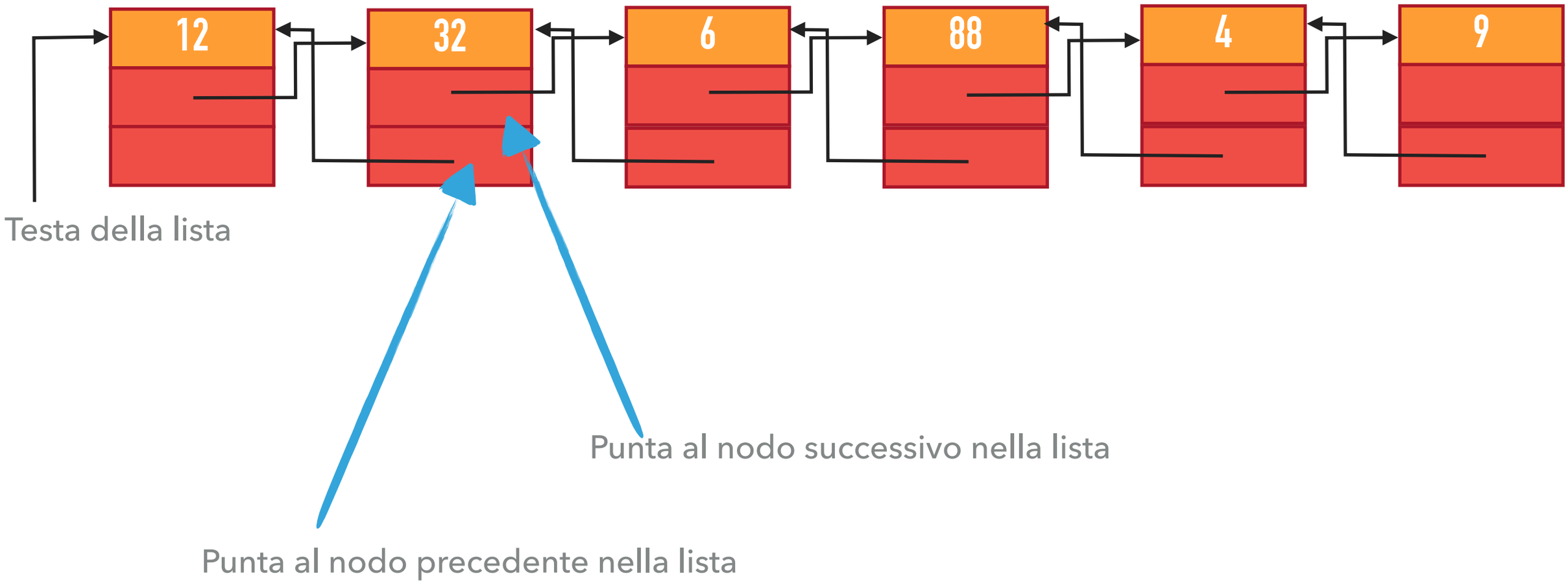
INFORMATICA

LISTA CONCATENATA DOPPIA

- Abbiamo visto che possiamo ottenere un migliore costo di inserimento usando una lista concatenata singola
- Però abbiamo anche visto che i riferimenti al nodo successivo ci permettono di muoversi solo “in avanti”
- Questo ha conseguenze negative quando dobbiamo cancellare un nodo dalla lista
- Possiamo migliorare?

LA LISTA CONCATENATA DOPPIA

Possiamo provare ad aggiungere un riferimento al nodo predecessore nella lista



LA LISTA CONCATENATA DOPPIA. PSEUDOCODICE

```
class Nodo2:
   def __init__(self, value, succ, pred):
      self.value = value
      self.succ = succ
      self.pred = pred
```

- Abbiamo un reference aggiuntivo per memorizzare quale è il nodo precedente

OPERAZIONI

- Come nel caso della lista concatenata singola, abbiamo diverse possibili operazioni:
  - Accesso ad un elemento dato l'indice
  - Ricerca di un elemento
  - Inserimento
  - Rimozione

LA LISTA CONCATENATA DOPPIA: ACCESSO E RICERCA

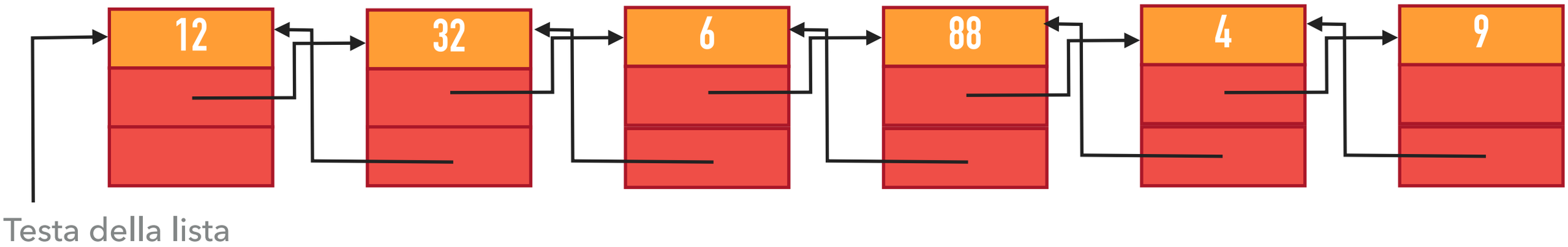
- L'accesso e la ricerca di un elemento sono esattamente gli stessi che con la lista concatenata singola
- Seguiamo solamente in riferimento al nodo successore
- Quindi abbiamo la stessa complessità:
  - $O(n)$ per accesso dato l'indice
  - $O(n)$ per ricerca

LA LISTA CONCATENATA DOPPIA: INSERIMENTO IN TESTA

- L'idea dell'inserimento in testa rimane simile
- Dobbiamo però gestire la sistemazione del riferimento al nodo predecessore nel caso la lista non sia vuota
- Il nodo che prima era in testa (che dopo l'inserimento è in seconda posizione) deve avere il riferimento al nuovo nodo come suo nodo predecessore

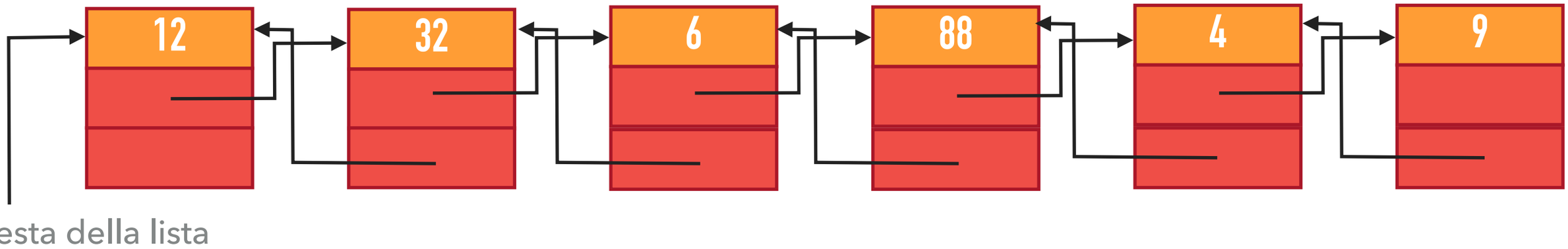
LA LISTA CONCATENATA DOPPIA: INSERIMENTO IN TESTA

Aggiungiamo un nuovo nodo con valore 7

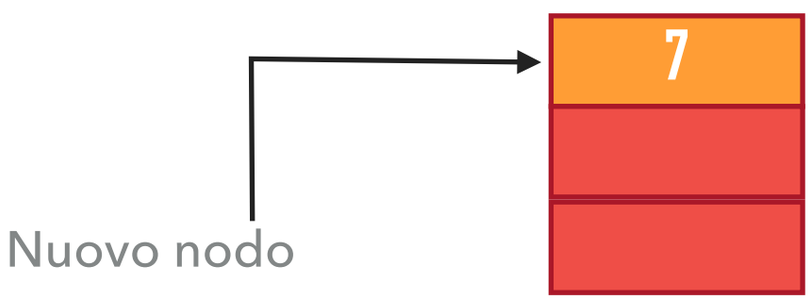


LA LISTA CONCATENATA DOPPIA: INSERIMENTO IN TESTA

Aggiungiamo un nuovo nodo con valore 7

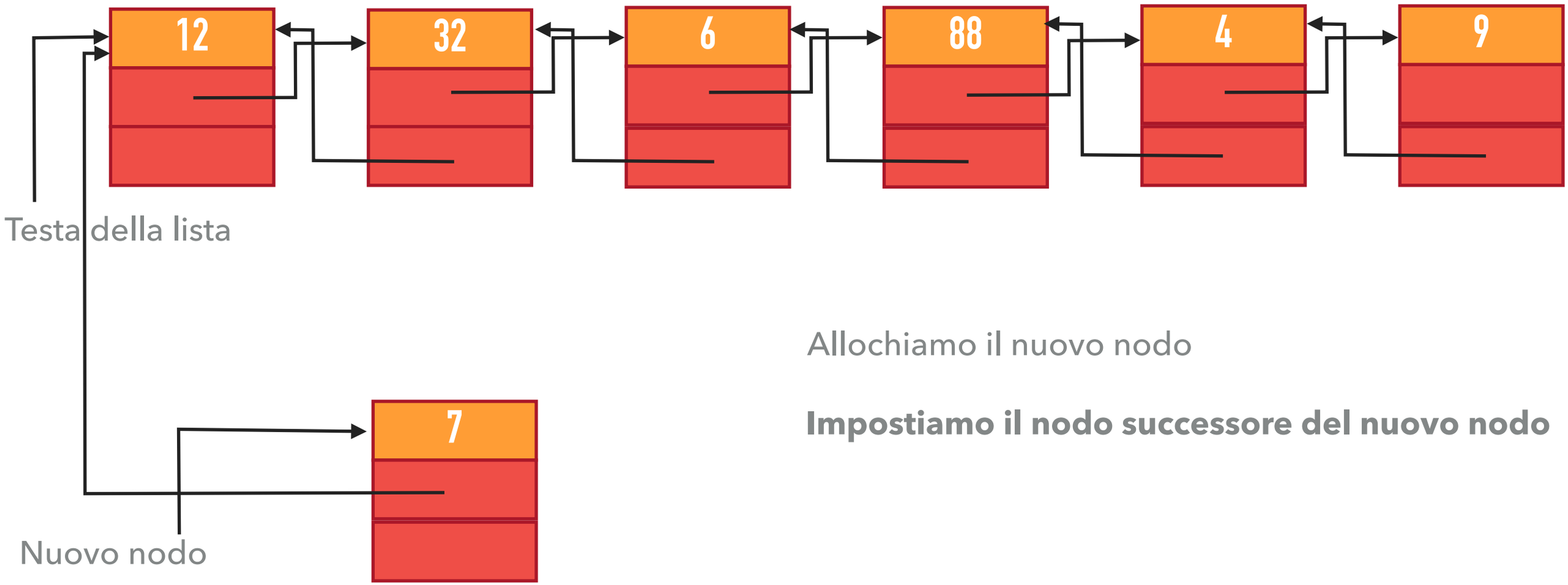


Allochiamo il nuovo nodo



LA LISTA CONCATENATA DOPPIA: INSERIMENTO IN TESTA

Aggiungiamo un nuovo nodo con valore 7

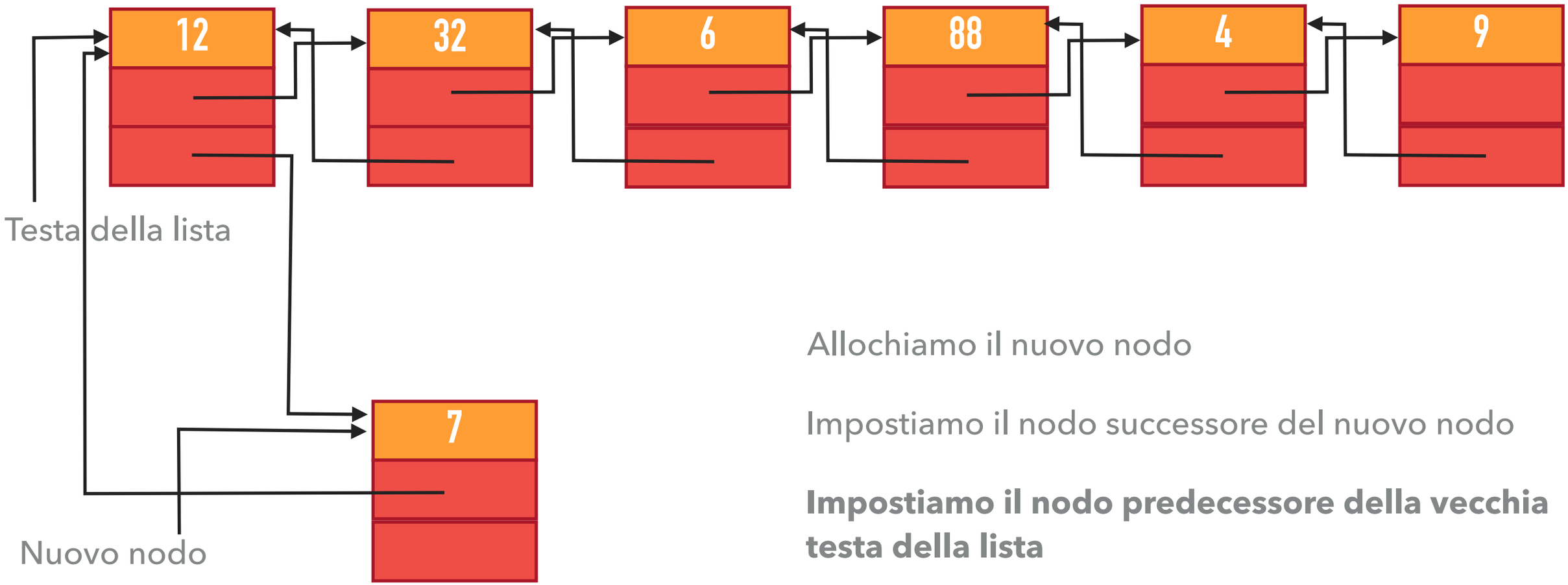


Allochiamo il nuovo nodo

Impostiamo il nodo successore del nuovo nodo

LA LISTA CONCATENATA DOPPIA: INSERIMENTO IN TESTA

Aggiungiamo un nuovo nodo con valore 7



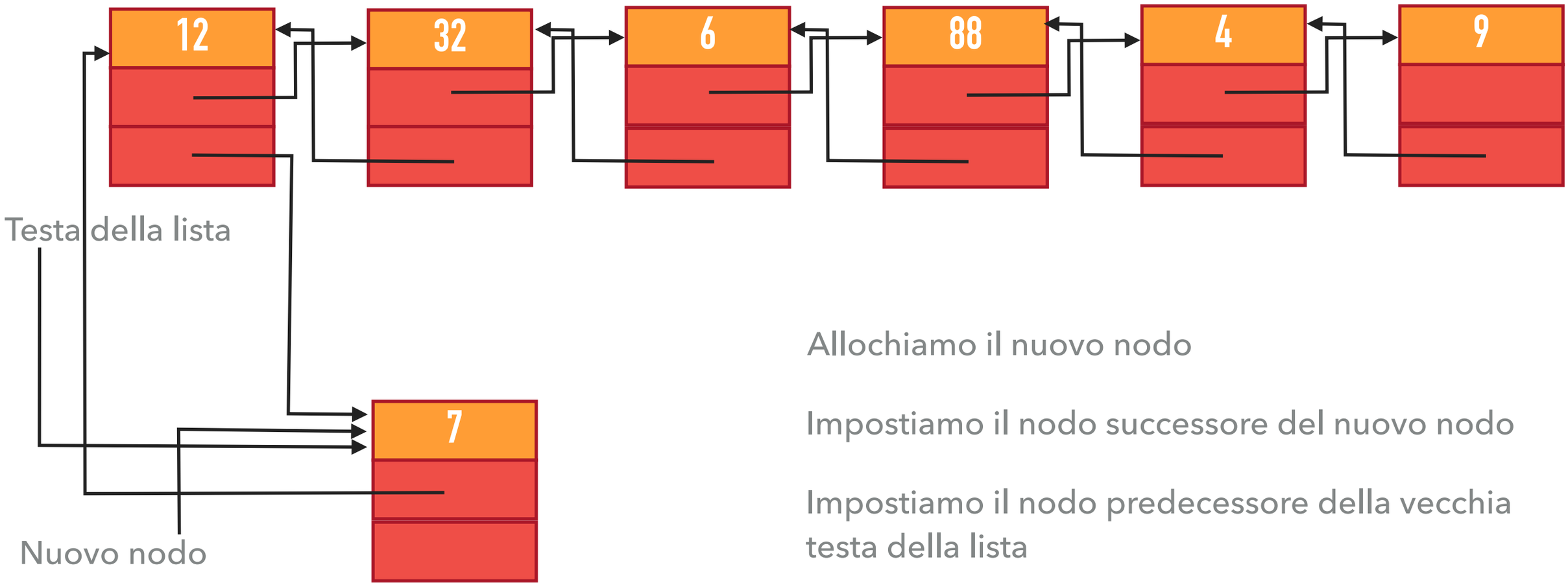
Allochiamo il nuovo nodo

Impostiamo il nodo successore del nuovo nodo

Impostiamo il nodo predecessore della vecchia testa della lista

LA LISTA CONCATENATA DOPPIA: INSERIMENTO IN TESTA

Aggiungiamo un nuovo nodo con valore 7



Allochiamo il nuovo nodo

Impostiamo il nodo successore del nuovo nodo

Impostiamo il nodo predecessore della vecchia testa della lista

Aggiorniamo la testa della lista

LA LISTA CONCATENATA DOPPIA: INSERIMENTO IN TESTA

```
Parametri: testa (reference al primo elemento), x (valore da inserire)
nuovo_nodo = alloca un nuovo nodo
nuovo_nodo.value = x
nuovo_nodo.succ = testa
nuovo_nodo.prec = None
if testa ≠ None # dobbiamo aggiustare il riferimento solo se esiste!
      testa.pres = nuovo_nodo
return nuovo_nodo
```

- Eseguiamo un numero costante di istruzioni, quindi il costo rimane $\Theta(1)$
- Esattamente come nel caso della lista concatenata singola

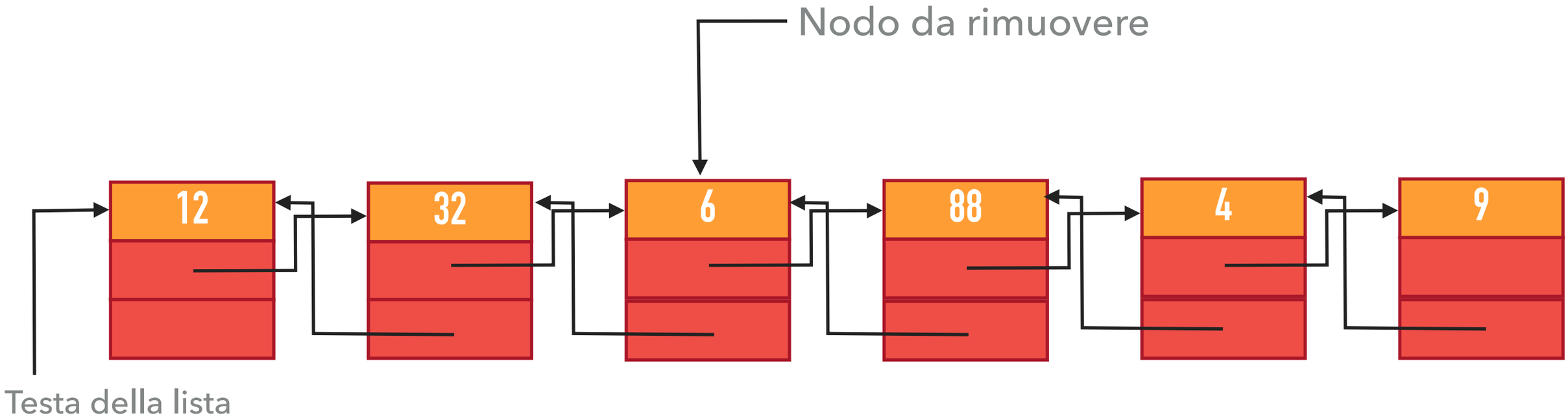
LA LISTA CONCATENATA DOPPIA: INSERIMENTO

- Per l'inserimento la procedura è simile a quella dell'inserimento in testa
- Semplicemente dobbiamo prima trovare la posizione in cui inserire il nuovo nodo
- E aggiustare più riferimenti (sia per il nodo successore che per il predecessore)

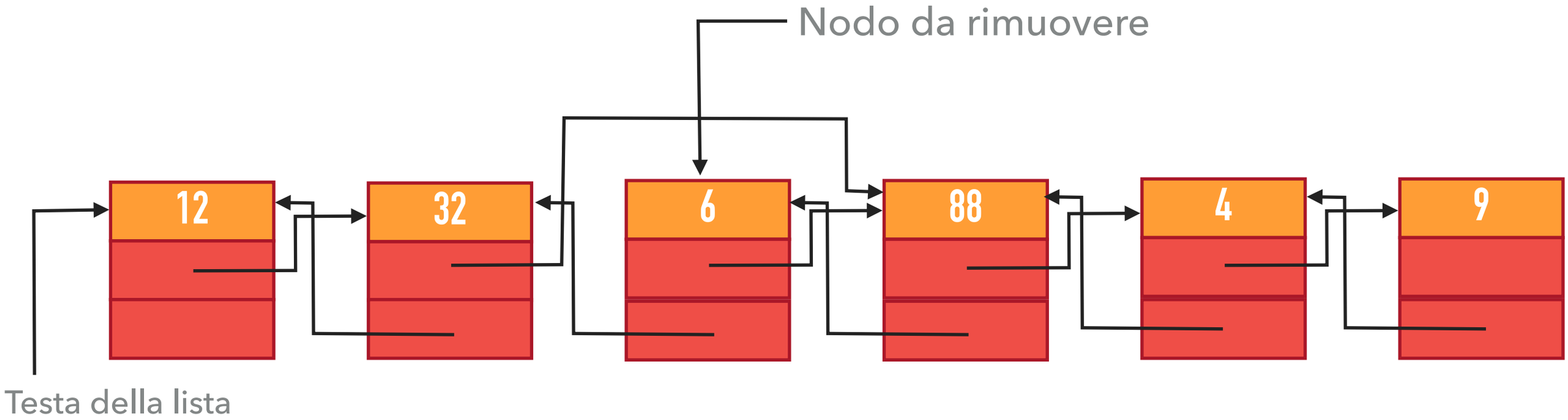
LA LISTA CONCATENATA DOPPIA: RIMOZIONE

- Qui c'è il primo cambiamento: nel caso delle liste concatenate singole non potevamo trovare il nodo predecessore se non scorrendo l'intera lista
- Adesso possiamo trovarlo!

LA LISTA CONCATENATA DOPPIA: RIMOZIONE

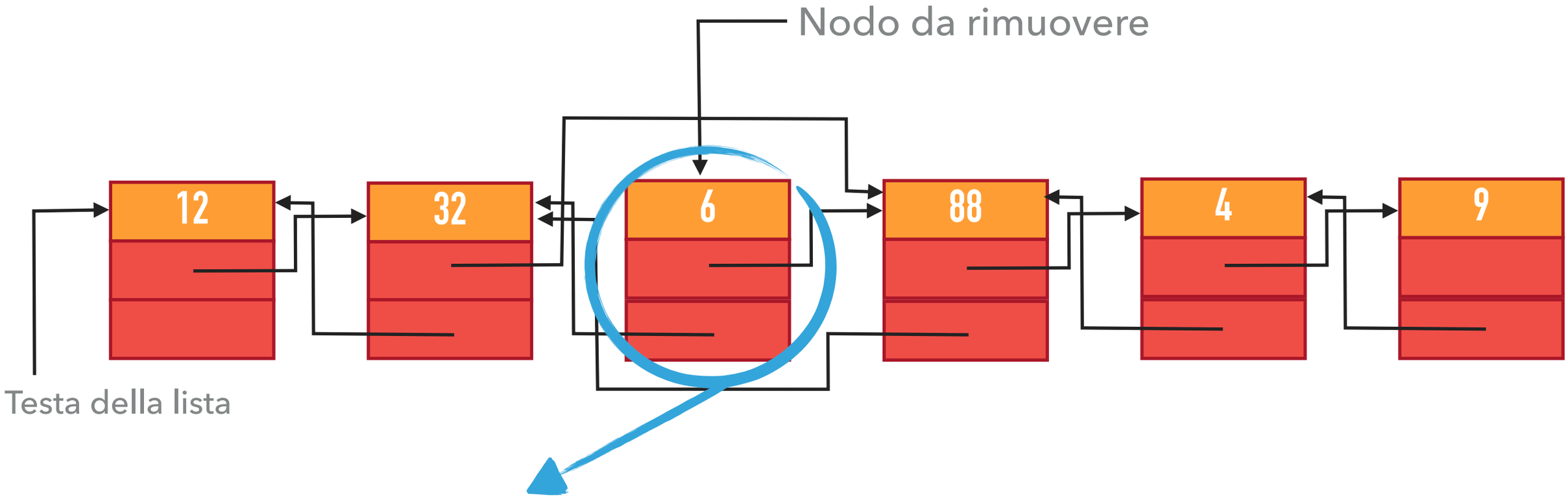


LA LISTA CONCATENATA DOPPIA: RIMOZIONE



Sistemiamo il riferimento del nodo predecessore

LA LISTA CONCATENATA DOPPIA: RIMOZIONE



Non abbiamo più nessuna freccia entrante nel nodo, lo abbiamo rimosso dalla lista

Sistemiamo il riferimento del nodo predecessore

Sistemiamo il riferimento del nodo successore

LA LISTA CONCATENATA DOPPIA: RIMOZIONE

```
Parametri: testa (reference al primo elemento)
           da_rimuovere (reference al nodo da rimuovere)
if da_rimuovere.prec ≠ None
     da_rimuovere.prec.succ = da_rimuovere.succ
else
     testa = da_rimuovere.succ
if da_rimuove.succ ≠ None
     da_rimuovere.succ.pres = da_rimuovere.prec
return testa
```

- Eseguiamo un numero costante di istruzioni, quindi il costo rimane $\Theta(1)$
- Quindi un miglioramento rispetto alla lista concatenata singola

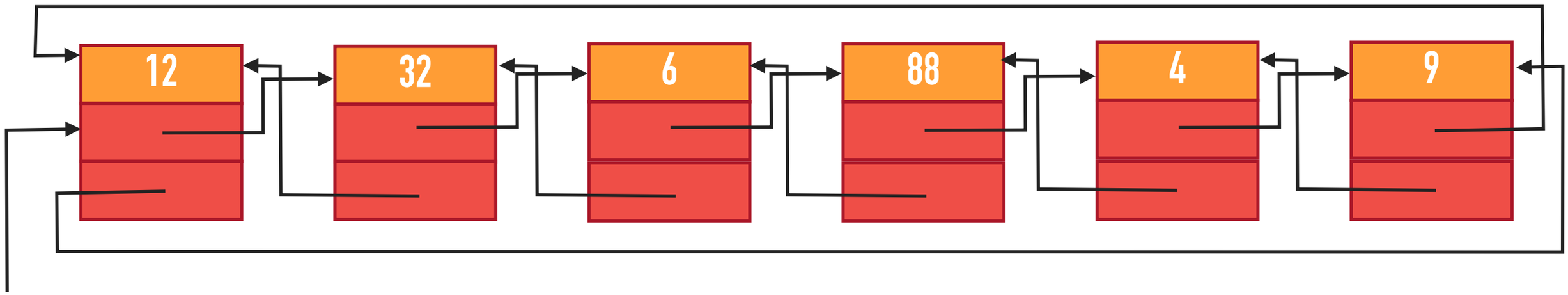
LA LISTA CONCATENATA DOPPIA: DISCUSSIONE

Operazione	Array	Lista concatenata singola	Lista concatenata doppia
Accedere ad un elemento	$O(1)$	$O(n)$	$O(n)$
Ricerca	$O(n)$	$O(n)$	$O(n)$
Inserimento	$O(n)$	$O(1)$ (se in testa)	$O(1)$ (se in testa)
Rimozione	$O(n)$	$O(n)$	$O(1)$

LA LISTA CONCATENATA: VARIANTI

- Una variante comune alla lista concatenata (doppia o singola) è rendere la lista circolare
- Questo significa "collegare" l'ultimo nodo della lista con il primo
- Attenzione: non possiamo più controllare quando la lista è terminata vedendo quando un reference è None (Null o nil in altri linguaggi)

LA LISTA CONCATENATA DOPPIA CIRCOLARE




```
nodo_corrente = testa
print(nodo_corrente.value)
nodo_corrente = nodo_corrente.succ
while nodo_corrente ≠ testa
   print(nodo_corrente.value)
   nodo_corrente = nodo_corrente.succ
```

Esempio di come effettuare la stampa degli elementi di una lista concatenata circolare.

Cosa succederebbe se verificassimo invece nodo_corrente ≠ None?